

Stack Overflow is a question and answer site for professional and enthusiast programmers. It's 100% free, no registration required.

Take the 2-minute tour ×

Serializing OpenCV Mat_<Vec3f>

I'm working on a robotics research project where I need to serialize 2D matrices of 3D points: basically each pixel is a 3-vector of floats. These pixels are saved in an OpenCV matrix, and they need to be sent over inter-process communication and saved into files to be processed on multiple computers. I'd like to serialize them in an *endian/architecture-independent, space-efficient* way, as quickly as possible.

cv::imencode [here](#) would be perfect, except that it only works on 8-bit and 16-bit elements, and we don't want to lose any precision. The files don't need to be human-readable (although we do that now to ensure data portability, and it's incredibly slow). Are there best practices for this, or elegant ways to do it?

Thanks!

c++ serialization opencv

asked Nov 13 '10 at 3:08



btown

1,035 8 13

- 1 I think it's not possible to have an endian-independent file format. You could have an "indicator" integer at the beginning of the file though (something like 0xFFFF). The position of the FF and FE will tell you the order to read the file. – [Utkarsh Sinha](#) Nov 15 '10 at 8:04
- 2 FYI if anyone comes across this post, our group is now using ROS and its built-in serialization for OpenCV matrices. It's incredibly performant, and can log uncompressed 640x480 at over 30fps on our hardware - more importantly, it's better tested than a roll-our-own solution would ever be. But the solutions presented here are awesome nonetheless! – [btown](#) Feb 25 '12 at 7:47

[add comment](#)

4 Answers

Edit: Christoph Heindl has commented on this post with a link to his blog where he has improved on this serialisation code. Highly recommended!

<http://cheind.wordpress.com/2011/12/06/serialization-of-cvmat-objects-using-boost/>

--

For whoever it may benefit: Some code to serialize Mat& with boost::serialization I haven't tested with multi-channel data, but everything should work fine.

```
#include <boost/serialization/vector.hpp>
```

```
BOOST_SERIALIZATION_SPLIT_FREE(Mat)
```

```
namespace boost {
```

```
namespace serialization {
```

```
    /** Mat */
```

```
    template<class Archive>
```

```
    void save(Archive & ar, const Mat& m, const unsigned int version)
```

```
    {
```

```
        size_t elemSize = m.elemSize(), elemType = m.type();
```

```
        ar & m.cols;
```

```
        ar & m.rows;
```

```
        ar & elemSize;
```

```
        ar & elemType; // element type.
```

```
        size_t dataSize = m.cols * m.rows * m.elemSize();
```

```
        //cout << "Writing matrix data rows, cols, elemSize, type, datasize: (" <<
```

```
        for (size_t dc = 0; dc < dataSize; ++dc) {
```

```

        ar & m.data[dc];
    }
}

template<class Archive>
void load(Archive & ar, Mat& m, const unsigned int version)
{
    int cols, rows;
    size_t elemSize, elemType;

    ar & cols;
    ar & rows;
    ar & elemSize;
    ar & elemType;

```

Now, mat can be serialized and deserialized as following:

```

void saveMat(Mat& m, string filename) {
    ofstream ofs(filename.c_str());
    boost::archive::binary_oarchive oa(ofs);
    //boost::archive::text_oarchive oa(ofs);
    oa << m;
}


void loadMat(Mat& m, string filename) {
    std::ifstream ifs(filename.c_str());
    boost::archive::binary_iarchive ia(ifs);
    //boost::archive::text_iarchive ia(ifs);
    ia >> m;
}

```

I've used the binary_oarchive and binary_iarchive here to keep the memory usage down. The binary format doesn't provide portability between platforms, but if desired the text_oarchive/iarchive can be used.

edited Dec 19 '11 at 12:49

answered Jun 10 '11 at 20:46

 [TumbleCow](#)
1,904 2 11 22

2 two improvements: you could use `boost::serialization::make_array` instead of a custom for-loop to save individual matrix elements and loss-less zlib compression can be easily added using `boost.iostreams`. I've compiled a [post](#) providing full source. – [Christoph Heindl](#) Dec 6 '11 at 8:54

Great post, thanks for sharing the improvements back! - I've added a link to the top of this post to make sure that people find it. – [TumbleCow](#) Dec 19 '11 at 12:51

You're welcome! – [Christoph Heindl](#) Dec 19 '11 at 19:08

This won't work for non-continuous matrices, and you don't actually need to serialize `elem_size` because it's derived from the `type` of the matrix. – [user1520427](#) Jan 29 at 23:20

[add comment](#)

You could use `boost::serialization` for that. It's heavily optimized and is pretty easy to integrate.

Possible speed-ups for your case include serializing each object as a raw binary block (see `boost::serialization::make_binary`) and disabling version tracking (`BOOST_SERIALIZATION_DISABLE_TRACKING`).

Also, you can experiment with adding compression into your serialization routines to save space (and time *in case of data that is easily compressable*). This can be implemented with `boost::iostreams`, for example.

answered Nov 14 '10 at 20:28



Yippie-Ki-Yay

3,578 7 35 92

thanks, I'll try it out! – btown Nov 15 '10 at 3:45

add comment

I was recently asking myself a similar question, though specifically I was trying to serialize opencv's `Mat` and `MatND` objects. Using `boost::serialize` is nice, but requires a couple tricks. As you don't want to go about modifying the internals of OpenCV itself to serialize these objects, you are forced to use what's called a "free" function. Since it is complicated to serialize the OpenCV objects, I found I was forced to split the serialize operation into save and load, each with a slightly different implementation. You need to use `boost/serialization/split_free.hpp` for this task. Boost provides good documentation for this here: http://www.boost.org/doc/libs/1_45_0/libs/serialization/doc/index.html.

Good luck!

answered Feb 2 '11 at 7:36



agodbehere

48 6

To ask an impolite question: Would you be willing to share your Mat serialization code? – TumbleCow Jun 6 '11 at 15:24

add comment

The earlier answers are good, but they won't work for non-continuous matrices which arise when you want to serialize regions of interest (among other things). Also, it is unnecessary to serialize `elemSize()` because this is derived from the `type` value.

Here's some code that will work regardless of continuity.

```
template<class Archive>
void serialize(Archive &ar, cv::Mat& mat, const unsigned int)
{
    int cols, rows, type;
    bool continuous;

    if (Archive::is_saving::value) {
        cols = mat.cols; rows = mat.rows; type = mat.type();
        continuous = mat.isContinuous();
    }

    ar & cols & rows & type & continuous;

    if (Archive::is_loading::value)
        mat.create(rows, cols, type);

    if (continuous) {
```

```
`const unsigned int` data_size = rows * cols * mat.elemSize();  
ar & boost::serialization::make_array(mat.ptr(), data_size);  
} else {  
    const unsigned int row_size = cols*mat.elemSize();  
    for (int i = 0; i < rows; i++) {  
        ar & boost::serialization::make_array(mat.ptr(i), row_size);  
    }  
}  
}
```

answered Jan 29 at 23:19



[user1520427](#)

575 4 16

[add comment](#)

Not the answer you're looking for? Browse other questions tagged [c++](#) [serialization](#)

[opencv](#) or [ask your own question](#).